SimSpark/SoccerServer RCSS as used for RoboNewbie

(based on the development of SimSpark and RoboNewbie in Summer 2012)

Hans-Dieter Burkhard and Monika Domanska
Humboldt-University Berlin, Institute of Informatics,
http://www.naoteamhumboldt.de/

The following material gives some additional information about the simulation software used in the RoboNewbie Project. Actually, the users of RoboNewbie can find all necessary information in the documents about Installation, How to Start, Quick Start Tutorial, and in the code documentation. Thus, the material presented here can serve as an additional source, and it also provides some more details.

It is mainly collected from the official Wiki provided on the web:
http://simspark.sourceforge.net/wiki/index.php/Main_Page (as in Summer 2012)

The Wiki describes the simulation software provided for the RoboCup competitions of the 3D simulation league. The software is free with open sources. Note that the simulator undergoes continuing evolutions by the RoboCup community for providing new challenges. Therefore, some information given here may become invalid in the future. At the same time the usage by RoboNewbie may change.

1. Overview

The overall system consists of the *SoccerServer* and the *agents*, i.e. the player programs.

The SoccerServer simulates the physical world: The playground, the ball and the bodies of the players according to the laws of physics. As parts of the body, the sensors and effectors of the players are simulated by the SoccerServer as well, see section 3: “The Nao-Model used by the SoccerServer”.

An agent is the “brain” of a player. It is an autonomous program to control the simulated body. The implementation of agents is explained separately.

The interaction between the SoccerServer and an agent is performed by messages which contain the sensations and action commands, respectively. The message formats are explained in section 4 “Communication between agents and SoccerServer”.

The system works cyclically with basic cycles of 20 msec:

1. The server sends individual server messages with sensations to the agents.
2. The agents can decide for new actions depending on their beliefs about the situation.
3. The agents can send their agent messages to the server for desired actions.
4. The server collects the agents messages and calculates the resulting new situation (poses and locations of the players, ball movement etc.) according to the laws of physics and the rules of the game.

Figure 1 “Interaction of the SoccerServer and an agent” gives an schematic view. Note that the message exchange is interleaved as explained in section 5: “Synchronization between Server and Agent”.

2. Simulation using SimSpark: The SoccerServer and the Monitor

The SoccerServer for the RoboNewbie Project can be downloaded as specified by the RoboNewbie installation document. It was originally down loaded from the Wiki in spring 2012 and compiled for Windows. Some parameters where changed as explained below.

The SoccerServer is started by calling rcssserver3d.exe in the folder simspark-svn-r300.

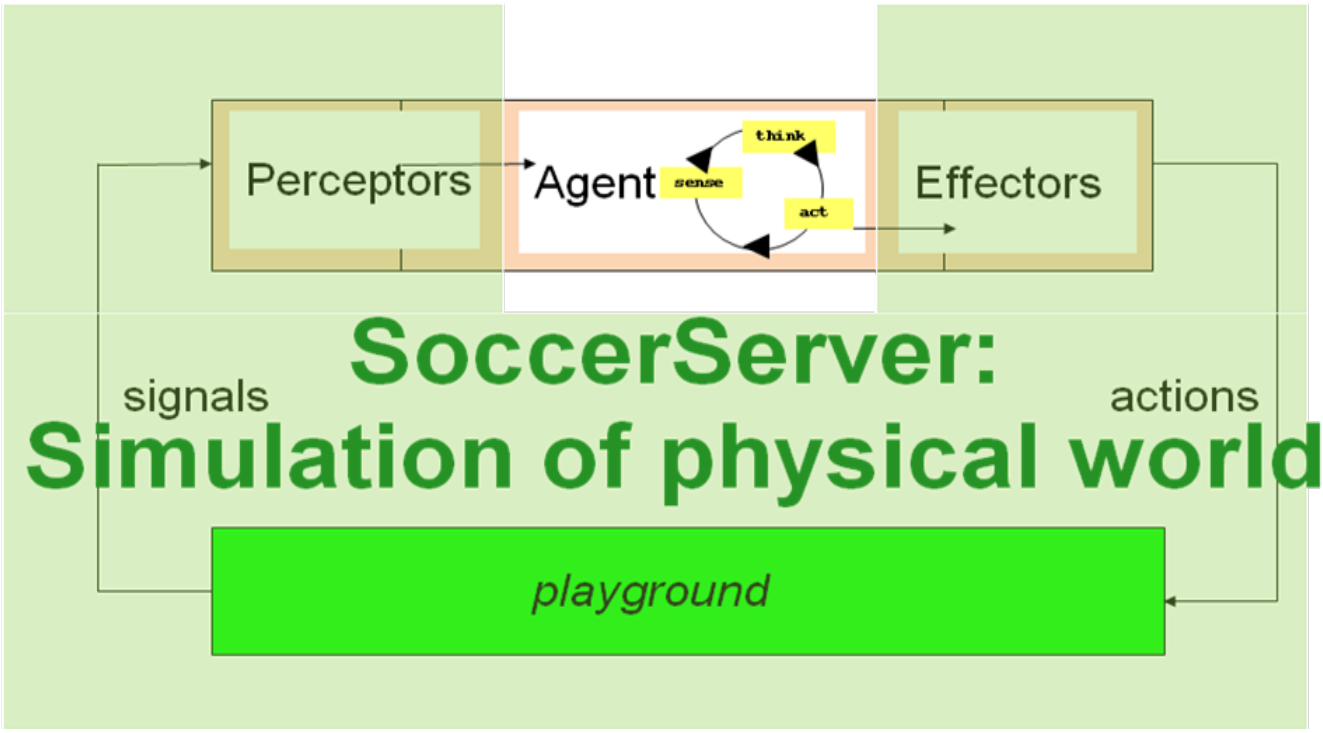

Figure 1: Interaction of the SoccerServer and an agent

It simulates the physical world for simulated soccer. It is based on SimSpark, a generic physical multi agent simulator system for agents in three-dimensional environments (http://simspark.sourceforge.net/wiki/). It uses the Open Dynamics Engine (ODE) for detecting collisions and for simulating rigid body dynamics. ODE allows accurate simulation of the physical properties of objects such as velocity, inertia and friction.

Besides the physical simulation, the simulator maintains the states of a soccer match according to the decisions of an automated referee. The referee decides about the game states according to the soccer rules of the RoboCup competitions (see section 7: “Running a game” for details). The server informs the agents about game states and prevents players from forbidden locations, e.g. crossing the halfway line before kick-off.

Simspark provides also a visualization: The SoccerMonitor (see section 6: “Monitor and User Interface” for details) visualizes the ongoing match on the playground. It is automatically started when calling rcssserver3d.exe . It serves as a user interface and allows for interventions by a human referee, especially for game start and interrupts (e.g. in case of game stuck).

Parameters of the simulator and the monitor can be changed by the various rb-files in the folder simspark-svn-r300.

The field coordinates have their center in the middle of the playground, the x-axis points to

the opponent goal. The size of the field in our distribution is 10x7 m (by a change in the configuration file naosoccersim.rb). It was changed to make life easier for beginners.

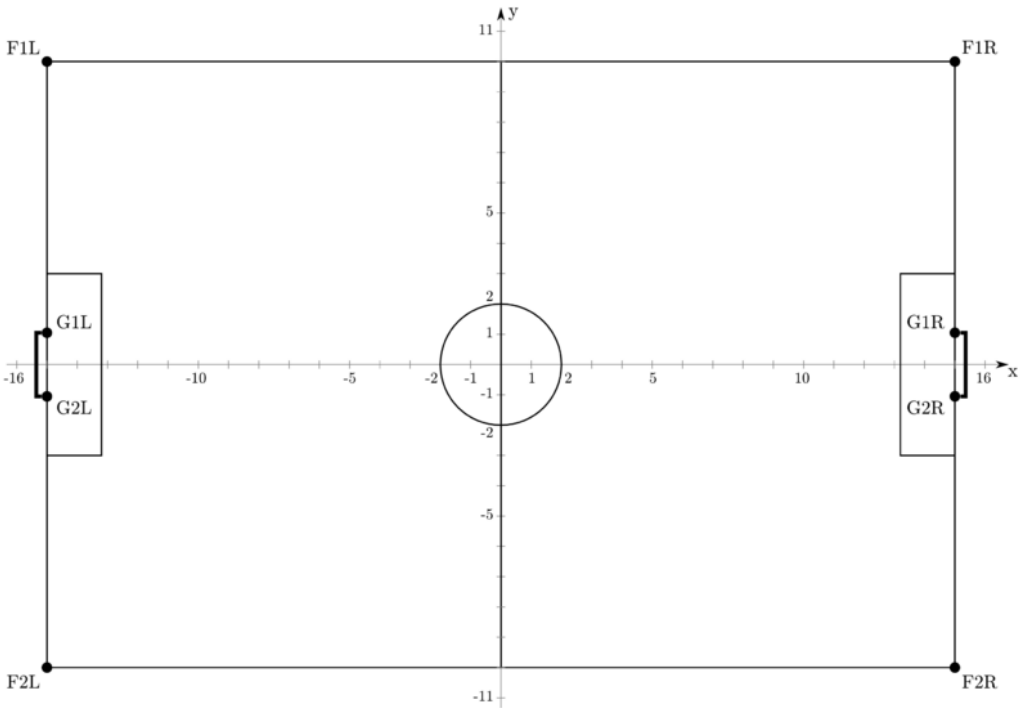

Figure 2: The Playground of the SoccerServer (from the Wiki). Note the different size (10x7m) in our distribution. The field parameters are set in the configuration file naosoccersim.rb)

3. The Nao-Model used by the SoccerServer

The simulated robot is based on the real robot Nao from the French company Aldebaran (cf. http://www.aldebaran-robotics.com). This robot is used in many scientific projects all over the world, it is also used in the Standard Platform League of RoboCup (cf. http://www.tzi.de/spl). Its height is about 57cm and its weight is around 4.5kg. Details of the physical properties are presented on the Wiki.

Figure 3: The real (left) and the virtual Nao (right) as shown in the Wiki

Actually, there are some differences between the real and the simulated robot. The simulated robot has 22 degrees of freedom, while the real one has only 21 (because the HipYawPitch joints are controlled by only one motor). The motors of the simulated robot can be controlled only by setting an angular speed, while the motors of the real robot are controlled via torque and stiffness. Not all sensors of the real robot are available by the simulated one. Moreover,

instead of the raw sensory data, the simulation provides preprocessed data in some cases (e.g. for the vision data). Further changes are under discussion in the RoboCup community.

The simulated robot has several effectors:

- Each joint can be controlled separately by related *hinge joint commands*. The figure "Joints of the Nao model" shows all joints of the simulated robot with their names and their identifiers.
- The *say effector* allows to communicate textual "voice" messages to other agents via the SoccerServer. Note that other communication between agents (e.g. via sockets) are not permitted by the rules.
- Further effectors are dedicated to inizialization (see below).

The robot is equipped with several perceptors. SimSpark uses the notion "perceptor" because some sensation messages contain preprocessed data ("percepts") instead of raw sensor data. The simulated robot has the following perceptors:

- Joint perceptors report the current angle of each joint.
- Gyroscope and accelerometer keep track of radial and axial movements of the upper torso in the three dimensional space.
- Force resistance perceptors in each foot indicate the actual pressure on it.
- The visual perceptor presents objects from preprocessed images of the camera at the head. The view range is 120 degrees horizontally, and 120 degrees vertically.
- The hear perceptor presents say messages from other players in textual format.
- The game state perceptor informs about the actual play time and play mode.

Details about the formats and contents of the messages are given below.

4. Communication between agents and SoccerServer

The communication between the SoccerServer and the agents is realized by message exchange using TCP (details are described in the Wiki). After starting, an agent must connect to the SoccerServer. The RoboNewbie agent is already prepared for communication, it connects just after its start. Then it has to send the initialization messages (see below).

For data transfer, the messages between the server and the agents are packed as byte streams. The messages use S-expressions ("symbolic expressions") as their basic data structure. S-expressions are either strings, or lists of simpler S-expressions. They can be easily parsed. Parsing is already implemented in the RoboNewbie agent which provides comfortable methods for handling the information exchange.

The agent interacts with the SoccerServer like a central control program communicates with sensors and effectors of a real robot.

Actually, some sensations are not presented as raw data but in an already preprocessed form as "percepts", and SimSpark uses the term "perceptor" instead of "sensor". All sensations of a single cycle are sent together as a server message which has to parsed for access to the information of sensors. Parsing and splitting to individual percepts are already performed by the RoboNewbie framework. Hence the user needs not to use the pure server messages. They are presented anyway below for understanding of further details like e.g. ranges of values.

Similarly, the effector commands of a single cycle are packed by the RoboNewbie agent to an agent message. Thereby, the agent message ends with a sync-message. This is necessary if the agent runs in sync mode (see "Synchronization between Server and Agents" below), while it is simply ignored in real time mode.

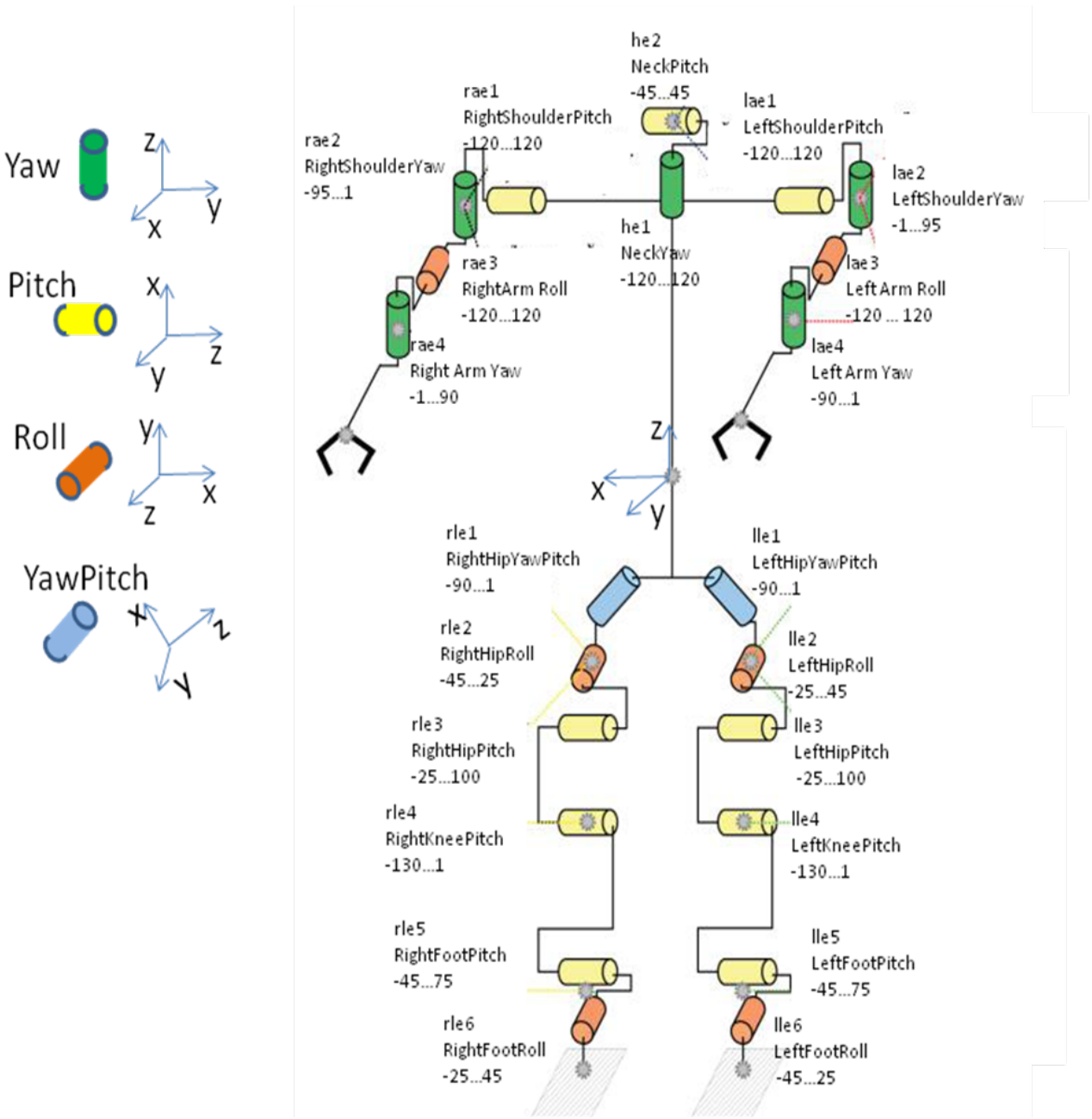

Figure 4: Joints of the Nao model (note that the model in the Wiki is not as accurate).

Joints revolve around the roles. Each joint has a readable name like RightShoulderPitch. Abbreviations like `rae2` are used as identifiers in the effector messages. The related perceptors are encoded with letter "j" instead of "e" (like `raj2`) in the perceptor messages. The ranges of the available angles are given below the names of the joints.

5. Communication between Agents and SoccerServer

The communication between the SoccerServer and the agents is realized by message exchange using TCP (details are described in the Wiki). After starting, an agent must connect to the SoccerServer. The RoboNewbie agent is already prepared for communication, it connects just after its start. Then it has to send the initialization messages (see below).

For data transfer, the messages between the server and the agents are packed as byte streams. The messages use S-expressions ("symbolic expressions") as their basic data structure. S-expressions are either strings, or lists of simpler S-expressions. They can be easily parsed. Parsing is already implemented in the RoboNewbie agent which provides comfortable methods for handling the information exchange.

The agent interacts with the SoccerServer like a central control program communicates with sensors and effectors of a real robot.

Actually, some sensations are not presented as raw data but in an already preprocessed form as "percepts", and SimSpark uses the term "perceptor" instead of "sensor". All sensations of a single cycle are sent together as a server message which has to parsed for access to the information of sensors. Parsing and splitting to individual percepts are already performed by the RoboNewbie framework. Hence the user needs not to use the pure server messages. They are presented anyway below for understanding of further details like e.g. ranges of values.

Similarly, the effector commands of a single cycle are packed by the RoboNewbie agent to an agent message. Thereby, the agent message ends with a sync-message. This is necessary if the agent runs in sync mode (see "Synchronization between Server and Agents" below), while it is simply ignored in real time mode.

Effector messages

- The joints have one degree of freedom as described figure 4 "Joints of the Nao model". The agent can control each of them by **hinge joint effector messages** of the format `(<joint> <speed>)` with a joint identifier and the angular speed in radians per second, e.g. `(rae 1)`for rotating the RightShoulder Pitch with a speed of about 60 degrees per second, i.e. about 1 degree per cycle.

  The server executes one command for each joint per cycle, i.e. the speed can be changed at each cycle. If no command is given, the movement continues with same speed as before. External forces like gravity may modify the actual speed, but not too much since the stiffness of the motors of the robot model is high. To stop a movement, the speed 0 has to be sent, and then the joint will remain in its position.

  Speed values can be executed by the server from -2π to 2π, i.e. one rotation per second at maximum. Values outside this range are pruned accordingly. Joints can be rotated in the ranges described in figure 4 "Joints of the Nao model". If the requested motion leads outside such a range, the joint will bounce until a new speed value is sent.

- The **say effector message** permits communication among agents by broadcasting messages. It has the format `(say <message>)` where message is some text with up to 20 characters from the ASCII printing character subset [0x20, 0x7E] except the white space character and the normal brackets (and). Not all say messages can be perceived (cf. the description of the hear perceptor message below). Example: `(say hello)`.

- The **create effector message** is used for the initialization after an agent has connected to the server. It has the format `(scene <filename>)`, where <filename> refers to a scene description file. It is used by the server to construct the physical representation and all further effectors and perceptors of the robot to be controlled by the agent. The initialized Nao robot in our distribution stands upright with zero angle at every joint. Example: `(scene rsg/agent/nao/nao.rsg).`

- The **init effector message** is sent once after the create effector message and serves also for initialization. The format is `(init (unum <playernumber>)(teamname <yourteamname>))`. It registers the agent as a member of the passed team with the passed number. Note that teamname cannot contain spaces. It is safe to use [A-Za-z_-] (regex character ranges).

  All players of one team have to use the same teamname and different playernumber values. If an agent sends 0 as playernumber, the number is assigned automatically by the server to the next free number. The side on which a team starts to play depends on which team connected first.

- The **beam effector message** allows a player to position itself on the field before the start of each half (in play mode BeforeKickOff). It has the format `(beam <x> <y> <rot>)`, where `x,y` are the coordinates of a position on the field, `rot` is the facing angle of the player in degrees (0 points to positive x-axis, 90 to positive y-axis of field coordinates). Example: `(beam 10.0 -10.0 0.0).`
- The **Synchronize Effector message** must be sent at the end of each simulation cycle if the simulation runs in sync mode (see section “Synchronization of Server and Agents” below). The server ignores this command if it is received in Real-Time Mode, so it is safe to use this message anyway as the last message in each cycle.

Perceptor messages

A server message comes at each cycle (each 20 msec) as collection of all recently available perceptor messages. An example for a single player on the playground before kick-off may look as follows:

```
(time (now 104.87))(GS (t 0.00) (pm BeforeKickOff))(GYR (n torso)
(rt 0.24 -0.05 0.02))(ACC (n torso) (a -0.01 0.05 9.80))(HJ (n hj1)
(ax -0.00))(HJ (n hj2) (ax -0.00))(See (G2R (pol 20.11 -18.92 0.84))
(G1R (pol 19.53 -13.04 0.90)) (F1R (pol 19.08 4.58 -1.54)) (F2R (pol
22.73 -33.49 -1.47)) (B (pol 10.12 -33.09 -2.94)) (L (pol 15.13 -
55.78 -2.03) (pol 8.67 10.24 -3.34)) (L (pol 22.78 -33.20 -1.23)
(pol 19.05 4.32 -1.76)) (L (pol 19.08 4.57 -1.55) (pol 1.81 60.14 -
17.11)) (L (pol 22.77 -33.23 -1.26) (pol 14.49 -59.60 -1.79)) (L
(pol 17.56 -11.77 -1.83) (pol 18.76 -23.38 -1.60)) (L (pol 17.58 -
11.67 -1.74) (pol 19.35 -10.53 -1.53)) (L (pol 18.71 -23.82 -1.97)
(pol 20.43 -21.36 -1.45)) (L (pol 11.68 -28.23 -2.73) (pol 10.93 -
23.90 -2.69)) (L (pol 10.91 -24.22 -2.95) (pol 9.84 -22.59 -3.02))
(L (pol 9.84 -22.64 -3.06) (pol 8.81 -25.74 -3.68)) (L (pol 8.83 -
25.33 -3.34) (pol 8.35 -32.24 -3.68)) (L (pol 8.35 -32.20 -3.64)
(pol 8.69 -39.32 -3.48)) (L (pol 8.68 -39.59 -3.71) (pol 9.63 -43.18
-3.37)) (L (pol 9.65 -42.85 -3.10) (pol 10.75 -42.17 -2.80)) (L (pol
10.75 -42.28 -2.89) (pol 11.61 -38.36 -2.50)) (L (pol 11.62 -38.15 -
2.33) (pol 11.94 -33.38 -2.58)) (L (pol 11.94 -33.31 -2.52) (pol
11.70 -28.03 -2.56)))(HJ (n raj1) (ax -0.00))(HJ (n raj2) (ax
0.00))(HJ (n raj3) (ax 0.00))(HJ (n raj4) (ax 0.00))(HJ (n laj1) (ax
```

```
-0.01))(HJ (n laj2) (ax 0.00))(HJ (n laj3) (ax -0.00))(HJ (n laj4)
(ax -0.00))(HJ (n rlj1) (ax 0.01))(HJ (n rlj2) (ax 0.00))(HJ (n
rlj3) (ax 0.01))(HJ (n rlj4) (ax -0.00))(HJ (n rlj5) (ax 0.00))(FRP
(n rf) (c -0.02 -0.00 -0.02) (f -0.02 -0.17 22.52))(HJ (n rlj6) (ax
-0.00))(HJ (n llj1) (ax -0.01))(HJ (n llj2) (ax 0.01))(HJ (n llj3)
(ax 0.00))(HJ (n llj4) (ax -0.00))(HJ (n llj5) (ax 0.00))(FRP (n lf)
(c 0.02 -0.01 -0.01) (f -0.08 -0.20 22.63))(HJ (n llj6) (ax 0.00))
```

The message starts with `(time (now <server time>))` where the number denotes the actual server time in seconds. It is expected to increase by 20 msec in each cycle, but sometimes its only 10 msec. It can also be more than 20 msec, especially in situations exceeding the capacity of the computer which runs the server.

After the time stamp, the server message contains the perceptor messages, whichcan be identified by related identifiers. There is no fixed ordering. If not stated otherwise below, the messages are sent every cycle and their values are not changed for simulated noise. But since only two decimal places are sent, there may be some small noise by rounding resp. truncation (the Wiki is not clear about that).

- **GameState perceptor message** `(GS (t <time>) (pm <playmode>))`
  It delivers the actual play time and play mode in each cycle. Play time starts from 0 at kickoff of the first half, and 300 at kickoff of the second half. The playmodes are described in section 7: “Running a game”.

- **HingeJoint perceptor message** `(HJ (n <name>) (ax <ax>))`
  Here `<name>` is the abbreviated name of the joint as given in figure 4 “Joints of the Nao model”, `<ax>` is the actual angle of the axis in degrees, e.g. `(HJ (n laj3) (ax -1.02))`.

- **ForceResistance perceptor message**
  `(FRP (n <name>) (c <px> <py> <pz>) (f <fx> <fy> <fz>))`
  Here `<name>` can be `lf` or `rf` for left or right feet, respectively. The sensors are located in the bottom of each foot. The first vector `<px> <py> <pz>` describes the point of origin in meters relative to the center of the sole of the foot, and the second vector `<fx> <fy> <fz>` the force on this point, respectively. The length of the force vector represents the given force in newton (kg m/s^2).
  The two vectors are just an approximation about the real applied force. The point of origin is calculated as weighted average of all contact points to which the force is applied, while the force vector represents the resulting force applied to all of these contact points. The information is just sent in case of a present collision of the corresponding body with another simulation object. If there is no force applied, the message of this perceptor is omitted.

- **Accelerometer perceptor message**`(ACC (n torso) (a <x> <y> <z>))`
  Here <x> <y> <z>is the current acceleration along the three axes of freedom of the center of the torso in m/s^2. It measures the acceleration relative to free fall, i.e. the accelerometer at rest will indicate approximately 1g (9.81m/s^2) upwards. To obtain the acceleration due to motion, this gravity offset should be subtracted.

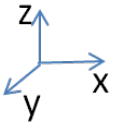

Figure 5: The orientation is given such that y points in the facing direction.

- **GyroRate perceptor message** `(GYR (n torso) (rt <x> <y> <z>))`
  The rotation angles `<x> <y> <z>` describe the change rates of the orientation of the center of the torso during the last cycle, i.e. the current angular velocities along the three axes of freedom of the torso in degrees per second. Refer to the Wiki concerning some open questions.

- **Vision perceptor message**

```
(See +(<name> (pol <distance> <angle1> <angle2>))
+(P (team <teamname>) (id <playerID>)
+(<bodypart> (pol <distance> <angle1> <angle2>)))
+(L (pol <distance> <angle1> <angle2>)
(pol <distance> <angle1> <angle2>)))
```

  The simulated camera is located in the geometrical center of the head. Its view range is each 120 degrees horizontally and vertically. The camera follows the movements of the head by the Neck Pitch and Neck Yaw angles, i.e. the orientation of the camera may be rotated with respect to the body coordinates of the robot.

  Vision messages are sent only at each third cycle, .i.e. every 60 msec. This corresponds to the lower vision frame rates in reality.

  In our distribution, the server sends a vision message with information about the position of seen objects on the playground. The related calculations by the server can be understood as a preprocessing of the image. Thereby, objects are not occluded by other objects.

  Positions of objects are given in spherical local coordinates `<distance> < horizontal angle> < vertical angle>` relative to the position and orientation of the camera. Angles are given in degrees, where positive horizontal angles are left and positive vertical angles are above the view direction.

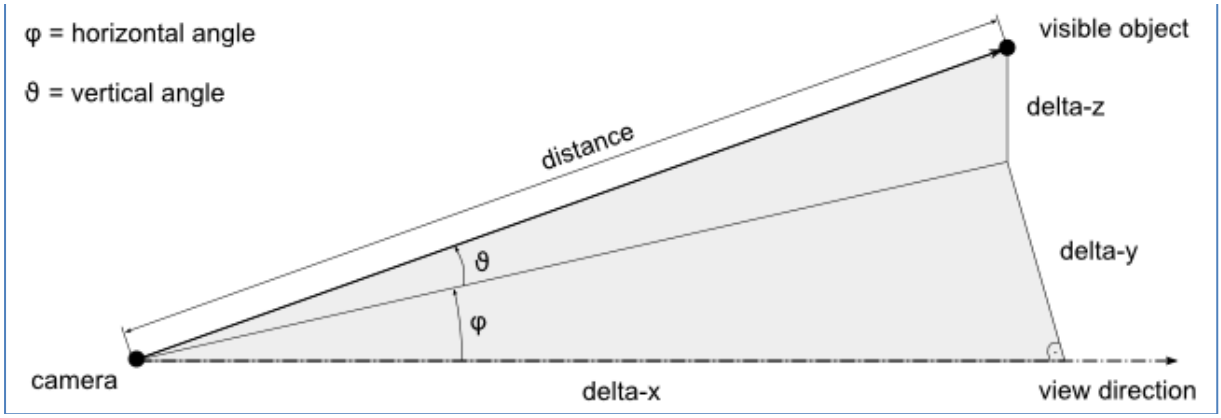

Figure 6: The facing angles of the vision perceptor.

To be closer to reality, some noise is added to the position values:

- A small calibration error is added to the camera position. For each axis the error is uniformly distributed between ±0.005m. The error is calculated once and remains constant during a simulation run.
- Dynamic noise normally distributed around 0.0 for each of:
  - Distance error: $\sigma^2 = 0.0965$ (also, distance error is multiplied by distance/100)
  - Horizonal angle (φ) error: $\sigma^2 = 0.1225$
  - Vertical angle (θ) error: $\sigma^2 = 0.1480$

Truncation to two decimal places results in an additional uniform error of up to 0.01. Fixed objects of the playground are landmarks and fieldlines. Landmarks are goalposts and corner points with identifiers as in the figure 2 “The Playground of the SoccerServer” (G1L and G1R for the own goal, G2L and G2R for the opponents goal, F1L, F1R, F2L, F2R for the corner flags). The coordinates of the points at the top of the goalposts are given (GoalHeight is specified as 0.8m in naosoccersim.rb), while the base points are given for the corner flags. Lines are non unique objects with the identifier “L”. Two position vectors ("start" and "end" point, bounded by the restricted view range) are given for each line.

Dynamic objects are the ball (identifier “B”) and other players (identified by “P”). The coordinates of the center of the ball are transmitted. For each player, the team name and the player number are given. Furthermore, the positions of body parts are given as far as they are in the view range (note that the robot may also see parts of its own body). Therewith, the pose of the robot can be estimated. The following body parts are regarded:

Visible body part	Identifier
Head	`head`
Right lower arm	`rlowerarm`
Left lower arm	`llowerarm`
Right foot	`rfoot`
Left foot	`lfoot`

Example of a see message:

```
(See (G2R (pol 17.55 -3.33 4.31))
(G1R (pol 17.52 3.27 4.07))
(F1R (pol 18.52 18.94 1.54))
(F2R (pol 18.52 -18.91 1.52))
(B (pol 8.51 -0.21 -0.17))
(P (team teamRed) (id 1)
(head (pol 16.98 -0.21 3.19))
(rlowerarm (pol 16.83 -0.06 2.80))
(llowerarm (pol 16.86 -0.36 3.10))
(rfoot (pol 17.00 0.29 1.68))
(lfoot (pol 16.95 -0.51 1.32)))
(P (team teamBlue) (id 3)
(rlowerarm (pol 0.18 -33.55 -20.16))
(llowerarm (pol 0.18 34.29 -19.80))))
(L (pol 12.11 -40.77 -2.40) (pol 12.95 -37.76 -2.41))
(L (pol 12.97 -37.56 -2.24) (pol 13.32 -32.98 -2.20))
```

- **Hear perceptor message** (`hear <time> self/<direction> <message>`)
  It receives messages produced by other players using the say effector. `<time>` is the simulation time at which the given message was heard, `<direction>` is the horizontal direction in degrees indicating where the sound originated, or “`self`” indicating that the player is hearing their own words, `<message>` is a sequence of up to 20 characters from the ASCII printing character subset [0x20, 0x7E] except the white space character and the normal brackets (and).

  The number of messages which can be heard at the same time is bounded. Each player has the maximal capacity of one heard message by a specific team every two simulation cycles (i.e. every 0.04 seconds per team). There are separately tracked capacities for both teams, because teams should not be able to block the hear perceptors of their opponents by shouting permanently. If multiple messages are spoken by members of a team within two simulation cycles, only one will be heard (the first to reach the server) and the rest will be discarded. Messages shouted by oneself will always be heard.

6. Synchronization between the Server and the agents

As already presented in the overview, the basic cycle has a length of 20 msec.

At each cycle, the server calculates the actual situation in dependence of the previous situation and the commands received from the agents. The calculation regards the physical laws and the implemented rules of soccer play. Furthermore, the server calculates the individual sensations for each agent according to the new situation including the pose of the agent. This information is sent to the agents by a server message at every cycle, but not all perceptors are available at each cycle: Most importantly, the vision information comes only at each third cycle.

At each cycle, an agent can process the server message and decide for the next actions. He can send related effector messages to the server at each cycle. In sync mode, the server waits for the agent messages of all agents until it starts to calculate the new situation. This results in a deadlock, if one agent does not send its messages. The sync mode can be switched on (off) by setting the flag agentSyncMode to true (false) in the configuration file spark.rb.

In Real Time mode (if sync mode is switched off), the server will not regard the agent messages which do not come in time. If an agent messages comes in a later cycle, it will be processed in that cycle.

Vice versa, the server will send a server message at every cycle, which remains in the message stream until it is read by the agent. If an agent misses to read a message in time, then there can be several server messages in the stream, and the synchronization can be lost.

The exchange of messages is interleaved as depicted in figure 7 “Synchronisation between SoccerServer and agent”. This corresponds to the time needed to process information in reality. Hence, an action command sent by the agent at cycle t will be processed by the server at cycle t+1 and the result can be observed by the agent not before cycle t+2. This must be regarded for controlling, i.e. the control needs an appropriate forethought.

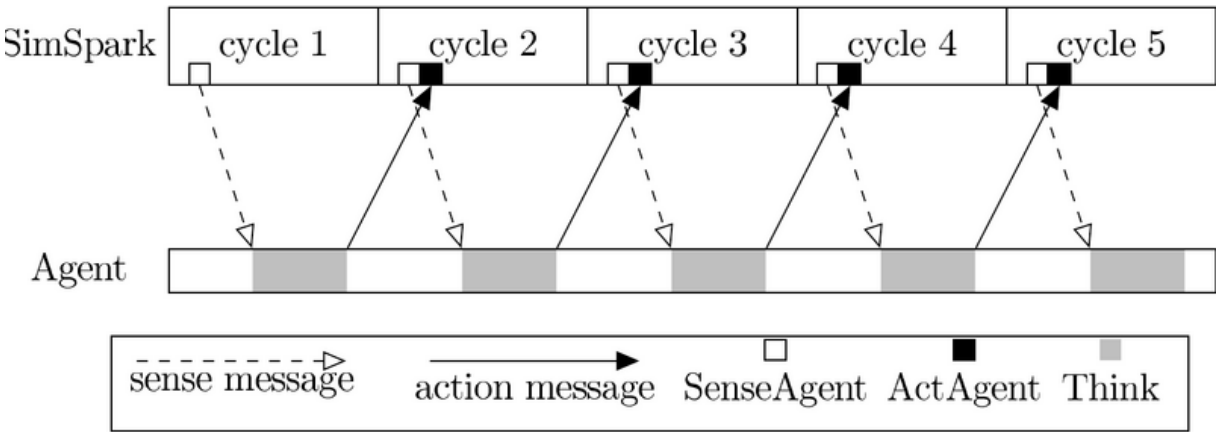

Figure 7: Synchronisation between SoccerServer and agent (from the Wiki)

7. Monitor and User Interface

The SimSpark monitor renders the current simulation it and displays the running time, the play mode, and the goal scores, respectively. An internal monitor is part of the SimSpark server and is started automatically with the server.

The monitor serves also as a user interface and accepts commands by key or mouse. These commands either control the movement of the monitor camera or send instructions back to the server as controls of the human referee.

Key	Function
q	quit monitor
left mouse	mouse look
pageup, keypad plus,right mouse	move camera up
pagedown, keypad minus	move camera down
a, left arrow	move camera left
d, right arrow	move camera right
w, up arrow	move camera forward
s, down arrow	move camera backward
1	camera to left goal
2	camera to left corner
3	camera to middle left
4	camera to middle right
5	camera to middle
6	camera to right corner
7	camera to right goal
l	free kick left
r	free kick right
k	kick off
b	drop ball
n	cycle selected agent
e	clear selection
lctrl+s	enter numeric agent selection mode (via l or r followed by a digit)
x	kill selected agent
m	move selected agent FreeKickDist meters back
p	pause the playback of a log file
f	move one step forward in the log file while paused
b	move one step backwards in the log file while paused
l	toggle forward/backward playback of log file

Some of the commands can be set only using the external monitor which is called by `rcssmonitor3d` (see the Wiki for details). It can be used also to replay logfiles of a simulation by the command `rcssmonitor3d --logfile logfilename.log`.

8. Running a game

The SoccerServer in the RoboNewbie Project is started by calling rcssserver3d.exe in the folder simspark-svn-r300. It automatically opens the internal monitor which shows the empty playground. Then the agent programs of both teams are started and positioned according to their beam effector commands. Team name and player number are specified by the agents in their init effector messages. The side on which a team starts to play depends on which team connected first.

The game is started by the human referee with the monitor-command “k” for kick-off. Then the game proceeds according to the implemented soccer rules or the monitor commands by a human referee, respectively. The soccer server (i.e. the built-in “referee”) observes the actions of the players and prevents them from certain illegal actions. The recent game state is shown on the monitor, and the game finishes if the regular time is over.

The version distributed for RoboNewbie has some modifications in the configuration file naosoccersim.rb, such that some restricting time periods are set to zero (the changes are marked there). To allow all players of all teams to start playing after kick-off without restrictions, use the monitor-command “b” (drop ball).

Play mode	Description	Conditions
`BeforeKickOff`	Before the match	The ball is at (0,0), the midfield and may not be moved. Players may use their beam effectors. Game time does not progress. This state is only left when a user instructs the simulator to start.
`KickOff_Left`	Kick off for the left team	The left team have a period in which to make their first kick. During this time the right team are not allowed to cross the centre line, or inside the centre circle. The left team may not cross the centre line, unless they are within the goal circle.
`KickOff_Right`	Kick off for the right team	The right team have a period in which to make their first kick. During this time the left team are lot allowed to cross the centre line, or inside the centre circle. The right team may not cross the centre line, unless they are within the goal circle.
`PlayOn`	Regular gameplay	
`KickIn_Left`	Kick in left team	The right team have kicked the ball off the side of the field. The ball is positioned on the sideline at the position it left the field. The right team are not allowed within a fixed radius of the ball, and the left team have a period of time in which to kick the ball back into play.
`KickIn_Right`	Kick in right team	The left team have kicked the ball off the side of the field. The ball is positioned on the sideline at the position it left the field. The left tem are not allowed within a fixed radius of the ball, and the right team have a period of time in which to kick the ball back into play.
`CORNER_KICK_LEFT`	Corner kick left team	

CORNER_KICK_RIGHT	Corner kick right team	
GOAL_KICK_LEFT	Goal kick for left team	
GOAL_KICK_RIGHT	Goal kick for right team	
GameOver	After the match	Play has finished. Agents may still move about, but no actions will have an effect upon the result of the match.
Goal_Left	Goal scored by the left team	This state exists for a few moments, before transitioning to KickOff_Right.
Goal_Right	Goal scored by the right team	This state exists for a few moments, before transitioning to KickOff_Left.
FREE_KICK_LEFT	Free kick for left team	The right team are not allowed within a fixed radius of the ball, and the left team have free access. PlayOn commences when the left team touches the ball, or if they fail to do so after a fixed period.
FREE_KICK_RIGHT	Free kick for right team	The left team are not allowed within a fixed radius of the ball, and the right team have free access. PlayOn commences when the right team touches the ball, or if they fail to do so after a fixed period.
NONE	No or unknown play mode	Agents should never receive this play mode.